CoreFMEE Handbook

v2.0

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 50200098-0

Release: March 2007

Trademarks

Table of Contents

Introduction

Core Overview

The CoreFMEE (Flash Memory Endurance Extender) macro uses the internal Flash memory blocks of Actel Fusion™ devices to emulate a serial EEPROM and to extend the life of the memory. It typically exists in a Fusion system between a microcontroller core and Flash memory. The serial EEPROM emulation allows users to reduce board-level components and save valuable board area by using a highly integrated design approach (internal Flash memory and FPGA logic). CoreFMEE has write protection modes for the Flash memory that can be controlled either by hardware or by software. Several sizes of emulated EEPROM devices are supported, from 128 bytes up to 2,048 bytes, and several physical-to-logical page mapping schemes are allowed, from 1 physical page per logic page up to 128 physical pages per logic page. CoreFMEE keeps track of when physical pages within the internal Flash memory blocks need to be used to extend the endurance of logical pages.

The block diagram for CoreFMEE is shown in Figure 1. CoreFMEE is broken into two functional blocks: a serial slave block that decodes serial commands synchronously to the user serial clock (SCL) and an NVM interface block that translates the decoded serial commands into Fusion Flash memory operations.

A typical application using the CoreFMEE macro is shown in Figure 2 on page 6. This typical application shows that CoreFMEE can work in conjunction with a user application that also accesses the internal Flash memory of a Fusion device, the arbitration of which is controlled by the user application.

CoreFMEE supports all devices in the Fusion family. Note that this handbook focuses on the operation of CoreFMEE and does not provide detail on the structure or behavior of the Fusion Flash memory. Refer to the *Fusion Family of Mixed-Signal FPGAs* datasheet for details on the Fusion Flash memory. Note that CoreFMEE has been designed to be used with an external device, but it could be adapted for use with user-created custom logic within the Fusion FPGA fabric.

CoreFMEE has five top-level parameters (Verilog) or generics (VHDL) used to configure the core. For a detailed description of the parameters and generics, refer to Table 2-1 on page 13.

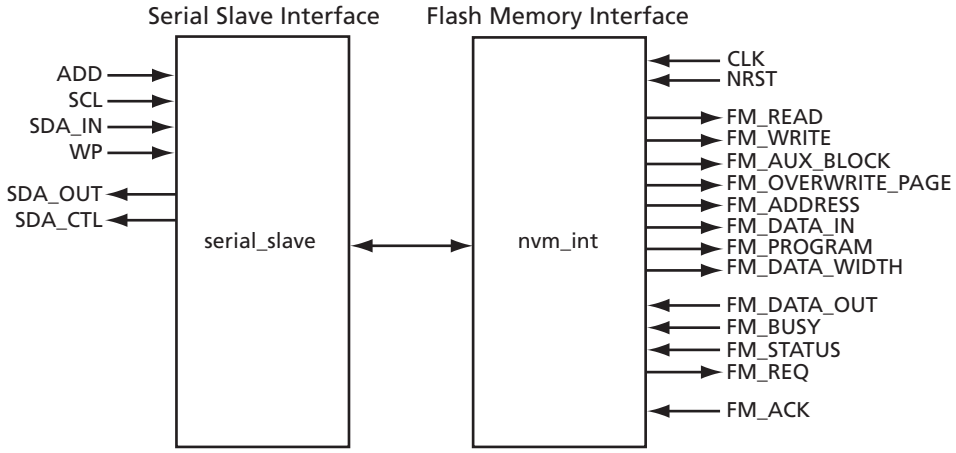


Figure 1 · CoreFMEE Block Diagram

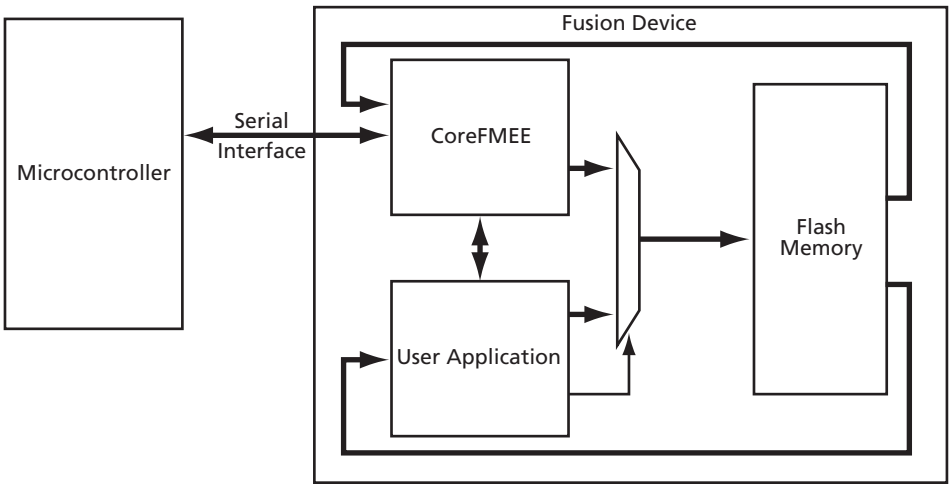


Figure 2 · CoreFMEE Typical Application

Device Utilization and Performance

CoreFMEE has been implemented in the Actel Fusion device family. A summary of the implementation data is listed in Table 1 through Table 3 on page 7.

Table 1 · CoreFMEE Device Utilization and Performance (minimum configuration)

Family	Tiles		Utilization		NVM Performance	Serial Performance
	Sequential	Combinatorial	Device	Total		
Fusion	94	251	AFS090-2	14%	88 MHz	5.5 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level generics/ parameters were set as follows: WP_MODE = 0, DEV_CONFIG = 0, ENDURANCE = 0, PAGE_MODE = 1, BASE_ADD = 0.

Table 2 · CoreFMEE Device Utilization and Performance (typical configuration)

Family	Tiles		Utilization		NVM Performance	Serial Performance
	Sequential	Combinatorial	Device	Total		
Fusion	117	420	AFS090-2	23%	56 MHz	3.5 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level generics/ parameters were set as follows: WP_MODE = 0, DEV_CONFIG = 2, ENDURANCE = 4, PAGE_MODE = 1, BASE_ADD = 0.

Table 3 · CoreFMEE Device Utilization and Performance (maximum configuration)

Family	Tiles		Utilization		NVM Performance	Serial Performance
	Sequential	Combinatorial	Device	Total		
Fusion	238	1,113	AFS090-2	60%	52 MHz	3.25 MHz

Note: Data in this table were achieved using typical synthesis and layout settings. Top-level generics/ parameters were set as follows: WP_MODE = 0, DEV_CONFIG = 4, ENDURANCE = 7, PAGE_MODE = 1, BASE_ADD = 0.

This core does not require the use of external device package pins, and it does not directly instantiate the Flash memory, though it does interface with the Flash memory, as shown in Figure 2 on page 6. The user instantiates the Flash memory either manually or within the Actel Libero® Integrated Design Environment (IDE). This macro has no specific electrical requirements and can be used in any of the Fusion devices.

1 Tool Flows

Licenses

CoreFMEE is licensed in three ways. Depending on your license type, tool flow functionality may be limited.

Evaluation

Pre-compiled simulation libraries are provided, allowing the core to be instantiated in CoreConsole and simulated within Libero IDE as described below. The design may not be synthesized as source code is not provided.

Obfuscated

Complete RTL code is provided for the core, allowing the core to be instantiated with CoreConsole and Simulation, Synthesis, and Layout to be performed within Libero IDE. The RTL code for the core is obfuscated,[1] and some of the testbench source files are not provided; they are pre-compiled into the compiled simulation library instead.

RTL

Complete RTL source code is provided for the core and testbenches.

CoreConsole

CoreFMEE is preinstalled in the CoreConsole IP Deployment Platform (IDP). To use the core,[2] simply drag it from the IP core list into the main window. The CoreConsole project can be exported to Libero IDE at this point, providing access just to CoreFMEE, or other IP blocks can be interconnected, allowing the complete system to be exported from CoreConsole to Libero IDE.

The core can then be configured using the configuration GUI within CoreConsole, as shown in Figure 1-1 on page 10, Figure 1-2 on page 10, and Figure 1-3 on page 11. Parameters can be entered manually within the CoreConsole GUI or imported from the SmartGen configuration file; the parameters are fully described in "Parameters" on page 13.

1. Obfuscated means the RTL source files have had formatting and comments removed, and all instance and net names have been replaced with random character sequences.

2. A CoreFMEE license is required to generate the design for export to Libero IDE for simulation and synthesis .

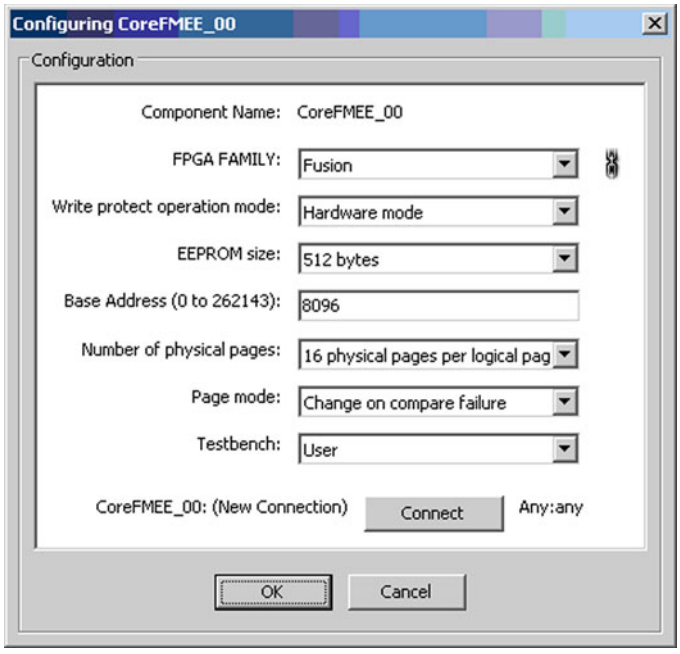


Figure 1-1 · CoreFMEE Configuration within CoreConsole

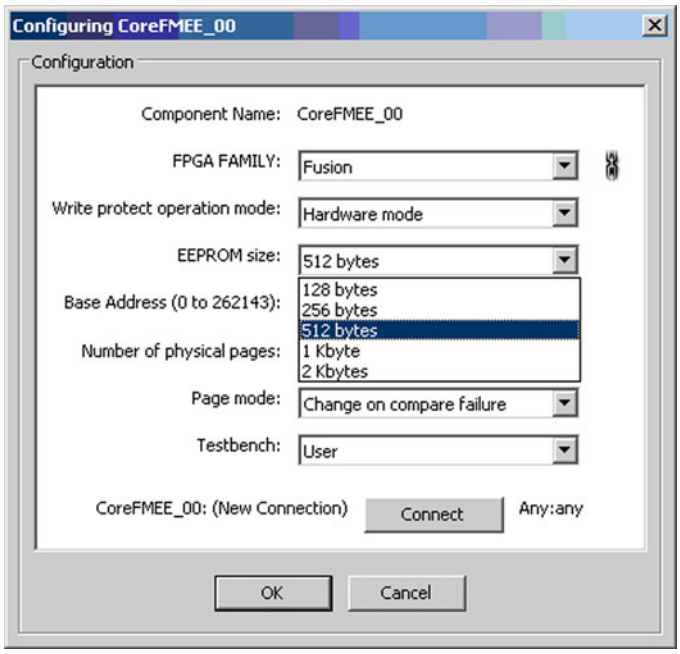


Figure 1-2 · CoreFMEE Configuration within CoreConsole (continued)

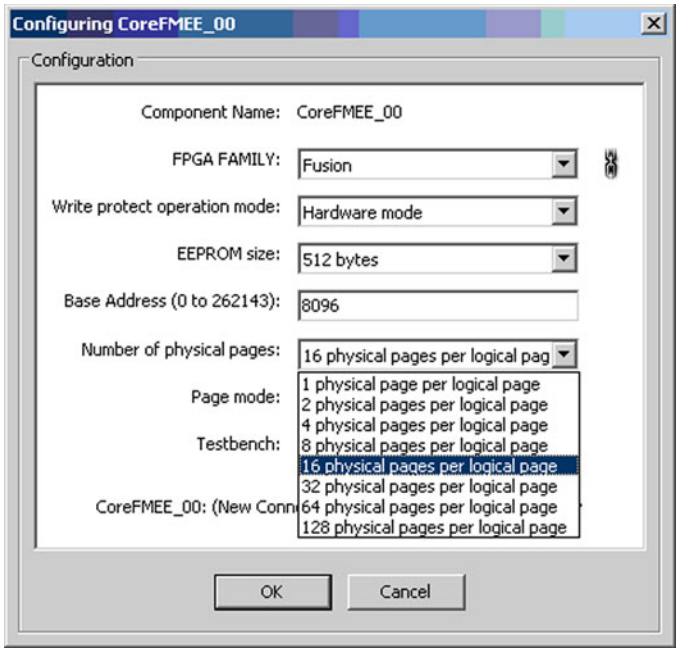


Figure 1-3 · CoreFMEE Configuration within CoreConsole (continued)

Importing into Libero IDE

After generating and exporting the core from CoreConsole, the core may be imported into Libero IDE. Create a new project in Libero IDE and import the CoreConsole project from the *LiberoExport* directory. Libero IDE will then install the core and the selected testbenches, along with constraints and documentation, into its project.

Note: If two or more DirectCores are required, they can both be included in the same CoreConsole project and imported into Libero IDE at the same time.

Simulation Flows

To run simulations, select the required testbench flow within CoreConsole and run **Save & Generate** from the Generate pane. The required testbench is selected through the Core Testbench Configuration GUI. Two simulation testbenches are supported with CoreFMEE:

- Simple CoreFMEE user application testbench (both VHDL and Verilog)
- Full CoreFMEE verification testbench (VHDL only)

When CoreConsole generates the Libero IDE project, it will install the appropriate testbench files. To run either the simple application or the full verification environment, simply set the design root to the CoreCFI instantiation in the Libero IDE design hierarchy and click the **Simulation** icon in the Libero IDE Design Flow window. This will invoke Model*Sim*® and automatically run the simulation.

Synthesis in Libero IDE

Set the design root appropriately and click the **Synthesis** icon in Libero IDE. The synthesis window appears, displaying the Synplicity® project. Set Synplicity to use the Verilog 2001 standard if Verilog is being used. To run Synthesis, click the **Run** icon.

Place-and-Route in Libero IDE

After setting the design root appropriately and running Synthesis, click the **Layout** icon in Libero IDE to invoke Designer. CoreFMEE requires no special place-and-route settings.

2 Interface Description

Parameters

CoreFMEE has parameters (Verilog) and generics (VHDL), described in Table 2-1, for configuring the RTL code. All parameters and generics are integer types.

Table 2-1 · CoreFMEE Parameter/Generic Descriptions

Name	Type	Description
FAMILY	Integer 0 to 99	Must be set to match the supported FPGA family: 17 – Fusion
WP_MODE	Integer 0 or 1	Determines which write protect operation is supported. 0 = Hardware mode 1 = Software mode
DEV_CONFIG	Integer 0 to 4	Indicates which size EEPROM is supported. 0 = 128 bytes (1 logical page) 1 = 256 bytes (2 logical pages) 2 = 512 bytes (4 logical pages) 3 = 1 kbyte (8 logical pages) 4 = 2 kbytes (16 logical pages)
BASE_ADD	Integer 0 to (2^{18} – 1)	Starting Fusion Flash memory address (sector, page) for CoreFMEE[1]
ENDURANCE	Integer 0 to 7	Number of physical pages available for each logical page[2] 0 = 1 physical page per logical page 1 = 2 physical pages per logical page 2 = 4 physical pages per logical page 3 = 8 physical pages per logical page 4 = 16 physical pages per logical page 5 = 32 physical pages per logical page 6 = 64 physical pages per logical page 7 = 128 physical pages per logical page
PAGE_MODE	Integer 0 or 1	Determines whether the change to a new physical page occurs from FM_STATUS[1:0] being over threshold or due to a compare failure. 0 = Change on overwrite threshold 1 = Change on compare failure

Notes:

1. *BASE_ADD must be set to a page boundary.*
2. *The more physical pages used, the longer the life of the Flash memory blocks is extended.*

Signals

The port signals for the CoreFMEE macro are defined in Table 2-2 on page 15 and shown in Figure 2-1. CoreFMEE has 57 I/O signals. This core is typically used with external device package pins (ADD[2:0], SDA, SCL, and WP as I/O pads—a total of six external I/Os). It does not directly instantiate the Flash memory, though it does interface with the Flash memory, as shown in Figure 2-1 (Flash memory interface signals begin with "FM_"). Note that if using SDA as a bidirectional signal at the top level of the FPGA design, the user will need to create the device SDA pin by instantiating tristate I/O pads using the SDA_CTL signal and the CoreFMEE SDA_IN and SDA_OUT signals. The user instantiates the Flash memory either manually or within Libero IDE.

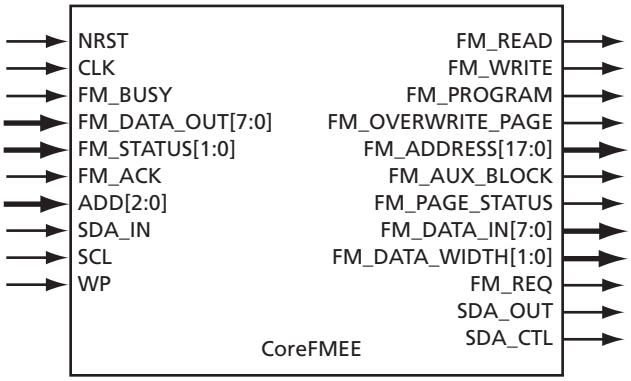


Figure 2-1 · CoreFMEE I/O Signal Diagram

Table 2-2 · CoreFMEE I/O Signal Descriptions

Name	Type	Description
NRST	Input	Active low asynchronous reset
CLK	Input	Flash memory interface clock—all Flash memory operations and status will be synchronous to the rising edge of this clock signal.
FM_READ	Output	When asserted, initiates a Flash memory read operation.
FM_WRITE	Output	When asserted, interface data present on FM_DATA_IN[7:0] is stored in the assembly buffer of the Flash memory.
FM_PROGRAM	Output	When asserted, this signal writes the contents of the assembly buffer into the cell array page addressed in the Flash memory within the Fusion device.
FM_OVERWRITE_PAGE	Output	When asserted, the page addressed is overwritten with the contents of the assembly buffer if the page is writeable.
FM_ADDRESS[17:0]	Output	The byte offset into the Flash memory cell array, assembly buffer, or data register
FM_AUX_BLOCK	Output	When this signal is asserted together with the FM_ADDRESS[17:0] signals, the contents of the auxiliary block of the Fusion Flash memory is being addressed.
FM_PAGE_STATUS	Output	When this signal is asserted during a read, it indicates that the status for the currently addressed page is being accessed.
FM_DATA_IN[7:0]	Output	Write data to the Fusion Flash memory.
FM_DATA_WIDTH[1:0]	Output	These output signals are used to select the data width mode of the Fusion Flash memory, and are fixed at '00' (byte mode). Only byte mode is used by CoreFMEE.
FM_BUSY	Input	Indicates that the Fusion Flash memory is performing an operation.
FM_DATA_OUT[7:0]	Input	Read data from the Fusion Flash memory
FM_STATUS[1:0]	Input	Status of the last operation completed
FM_REQ_	Output	Arbitration request for the Fusion Flash memory
FM_ACK	Input	Arbitration acknowledge from the Fusion Flash memory
ADD[2:0]	Input	Device address pins
SDA_IN	Input	Serial data input
SDA_OUT	Output	Serial data output
SDA_CTL	Output	This signal needs to be connected to the bidirectional control for the top-level I/O pad to which the SDA_IN input and SDA_OUT output are also connected. When this signal is logic 0, the SDA_OUT output is allowed to propagate out. When it is logic 1, the SDA_IN input is allowed to propagate in.
SCL	Input	Serial Clock Input—the serial_slave block is synchronous to this clock domain.
WP	Input	Write Protect—if hardware write protect is not used, this signal should be tied to logic 0.

Note: All signals are active high (logic 1) unless otherwise noted.

3 Functional Description

Flash Memory Endurance Algorithm

The logical-to-physical mapping of Flash memory pages in CoreFMEE utilizes a mechanism called linear/forward mapping, in which a logical page is mapped into *N* physical pages (where *N* is determined by the ENDURANCE generic/parameter). Physical pages are mapped to the logical pages based on a simple address calculation. The writes begin at the first physical page and will continue in this page until a Flash memory program failure is detected. When a failure is detected at the end of a Flash memory programming sequence, the next write/program will be to the next sequential physical page. This write sequence continues until all *N* physical pages for this logical page have failures when programming. If the user continues to perform writes after the last page has encountered the failure mechanism, CoreFMEE will continue to write and program but will no longer be able to guarantee accurate data content.

Mapping of logical addresses to the *N* physical Flash memory pages is done in the following fashion:

Physical base address = BASE_ADD + {Logical page number concatenated with the endurance offset}

CoreFMEE essentially shifts the logical page number left, depending on the setting of the ENDURANCE generic/parameter. For example, if ENDURANCE is set to 0, a shift will not occur; if ENDURANCE is set to 1, a shift of one bit place will occur; if ENDURANCE is set to 2, a shift of two bits will occur; etc.

Table 3-1 · Addressing Example

FM Address	EE Logical Page	EE Physical Page
0x00000	0	0
0x00080	0	1
0x00100	0	2
0x00180	0	3
0x00200	1	0
0x00280	1	1
0x00300	1	2
0x00380	1	3
0x00400	2	0
0x00480	2	1
0x00500	2	2
0x00580	2	3
0x00600	3	0
0x00680	3	1
0x00700	3	2
0x00780	3	3

Note: For this example BASE_ADDRESS = 0, DEV_CONFIG = 2, ENDURANCE = 2

Flash Memory Initialization

During initialization, CoreFMEE uses the four configuration generics/parameters: BASE_ADD, ENDURANCE, PAGE_MODE, and DEV_CONFIG. The initialization routine determines the read and write physical addresses for each logical Flash memory page. The number of logical pages is determined by the DEV_CONFIG generic/parameter. CoreFMEE supports from 1 to 16 logical pages.

Initialization will be performed when a LOW to HIGH transition occurs on the NRST input signal.

Starting at the BASE_ADD address, CoreFMEE will read all the locations of each page until it has determined the read pointer for the logical page. If the first physical page does not have the IN_USE[1] bit set in the user data, the read pointer is set to page 0. Data reads for a logical page with no IN_USE indicators will return whatever data is in physical page 0. If the physical page contains a valid IN_USE indicator, but there is a compare error or write threshold error (depending on the setting of the PAGE_MODE generic/parameter) during the read of the page, then the read pointer is set to this page address, but the write indicator shows the page is not available for writes. If the page contains a valid IN_USE indicator and there is no ECC or write threshold error, both the read and write indicators are loaded with the current page address.

When the read pointer and write indicator have been initialized for a logical page, the process is repeated for the next logical page until the last set of logical page pointers has been initialized. Refer to Figure 3-1 on page 19 for a flow diagram of the initialization process.

The write protect setting for CoreFMEE is determined during the read of logical page 0. If the IN_USE bit is set in the auxiliary block of the Flash memory, the write protect state is initialized.

1. *IN_USE is a status bit stored in the auxiliary block of the Fusion Flash memory and indicates that the current page is in use.*

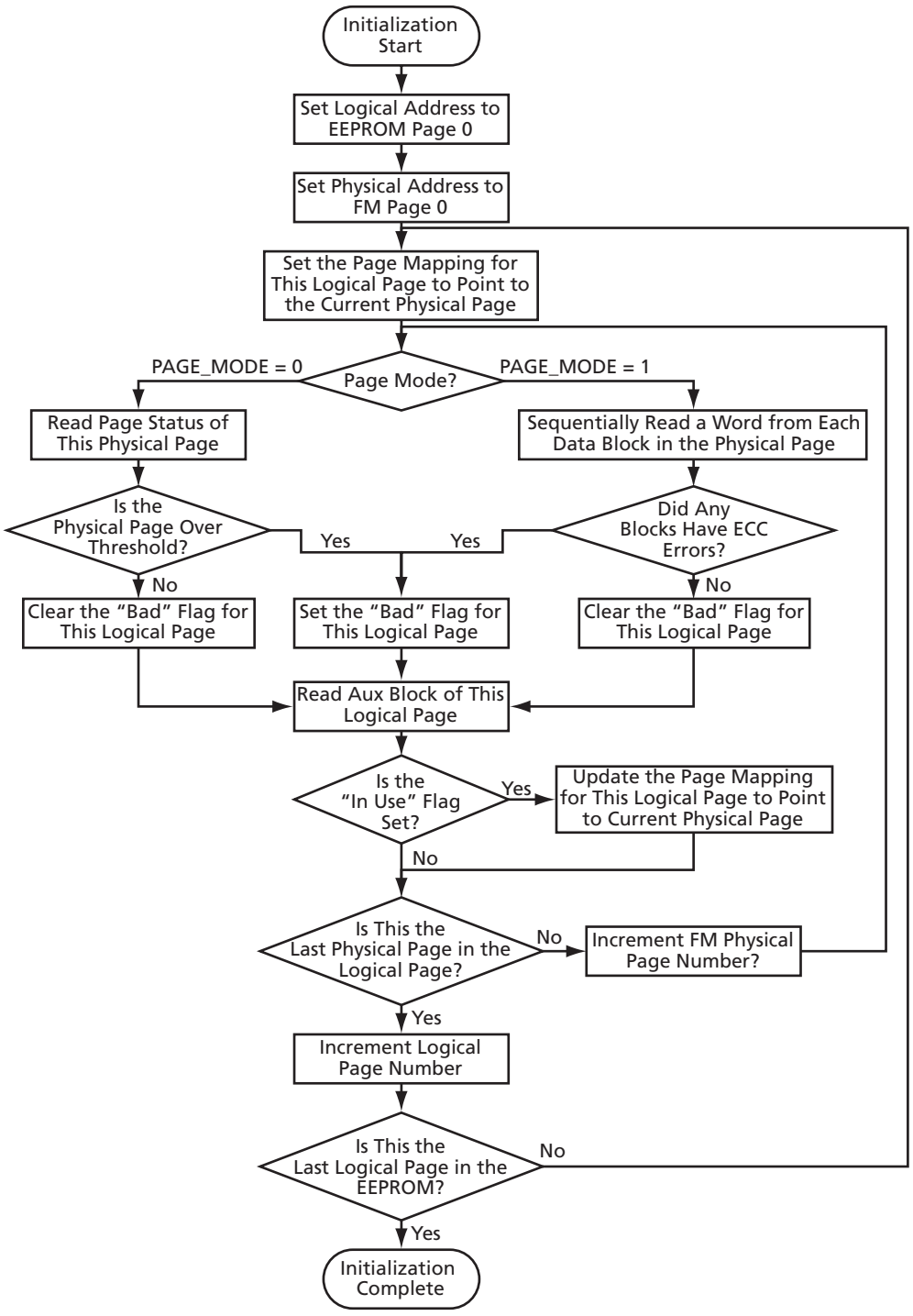


Figure 3-1 · Initialization Flow Diagram

Flash Memory Interface Operation

CoreFMEE utilizes the Fusion Flash memory commands listed in Table 3-2.

Table 3-2 · Fusion Flash Memory Commands Used

Command	How Command Is Used
READ	Utilized to read data for the serial interface. Used in conjunction with FM_AUX_BLOCK to read user data from the current page. Used in conjunction with FM_PAGE_STATUS to determine overwrite threshold during the initialization sequence.
WRITE	Used to write serial data to the Flash memory. Used with FM_AUX_BLOCK to write user data to the auxiliary block of the page.
PROGRAM	Used to program page to physical address after write to physical page. Used in conjunction with FM_OVERWRITE_PAGE to overwrite data in NVM to different page address when a failure occurs.

The COREFMEE_SERIAL_SLAVE block begins an EEPROM operation by sending a request to the COREFMEE_NVM_INT block. The COREFMEE_NVM_INT logic will then begin an arbitration phase where the output signal FM_REQ is set. Upon receipt of the FM_ACK input signal, it is passed along to the COREFMEE_SERIAL_SLAVE block as an acknowledge. The COREFMEE_NVM_INT block will hold the FM_REQ active until either the read is complete or the program is complete after the write or protect operation is complete.

When a read or write operation is initiated from the COREFMEE_SERIAL_SLAVE block, the COREFMEE_NVM_INT block will perform the logical-to-physical translation of the requested address. The COREFMEE_NVM_INT block will then perform a read/write operation on the Flash memory within the Fusion device. For read operations, the data from the Flash memory will be registered and presented to the COREFMEE_SERIAL_SLAVE block when the FM_BUSY input is no longer active. For write operations, the FM_DATA_IN[7:0] outputs will present the data byte as received and registered by the COREFMEE_SERIAL_SLAVE block.

The COREFMEE_NVM_INT block will perform a program operation to the Flash memory under two conditions: at the completion of write operations by the COREFMEE_SERIAL_SLAVE block and with the setting of serial protection logic by the COREFMEE_SERIAL_SLAVE block. When a program sequence is started, the COREFMEE_NVM_INT block will perform a write to the USER_DATA field in the auxiliary block of the Fusion Flash memory to set the IN_USE bit and to store WRITE_PROTECT information. The COREFMEE_NVM_INT block will then assert the FM_PROGRAM output signal and start a program operation. After completion of the program sequence, the COREFMEE_NVM_INT block will monitor the FM_STATUS[1:0] inputs. If the FM_STATUS[1:0] inputs are equal to '10' or '11' (depending on PAGE_MODE), indicating a program issue, the COREFMEE_NVM_INT block will store this information for the next program operation. If there was an issue on the previous program operation, the COREFMEE_NVM_INT block will perform the writes to the current physical page, but will change the FM_ADDRESS[17:0] outputs to the new physical page and assert the FM_PROGRAM and FM_OVERWRITE_PAGE signals. If the current physical page is the last physical page for the logical page, the current page will be programmed again.

Refer to Figure 3-2 for a high-level overview flow diagram of the write process.

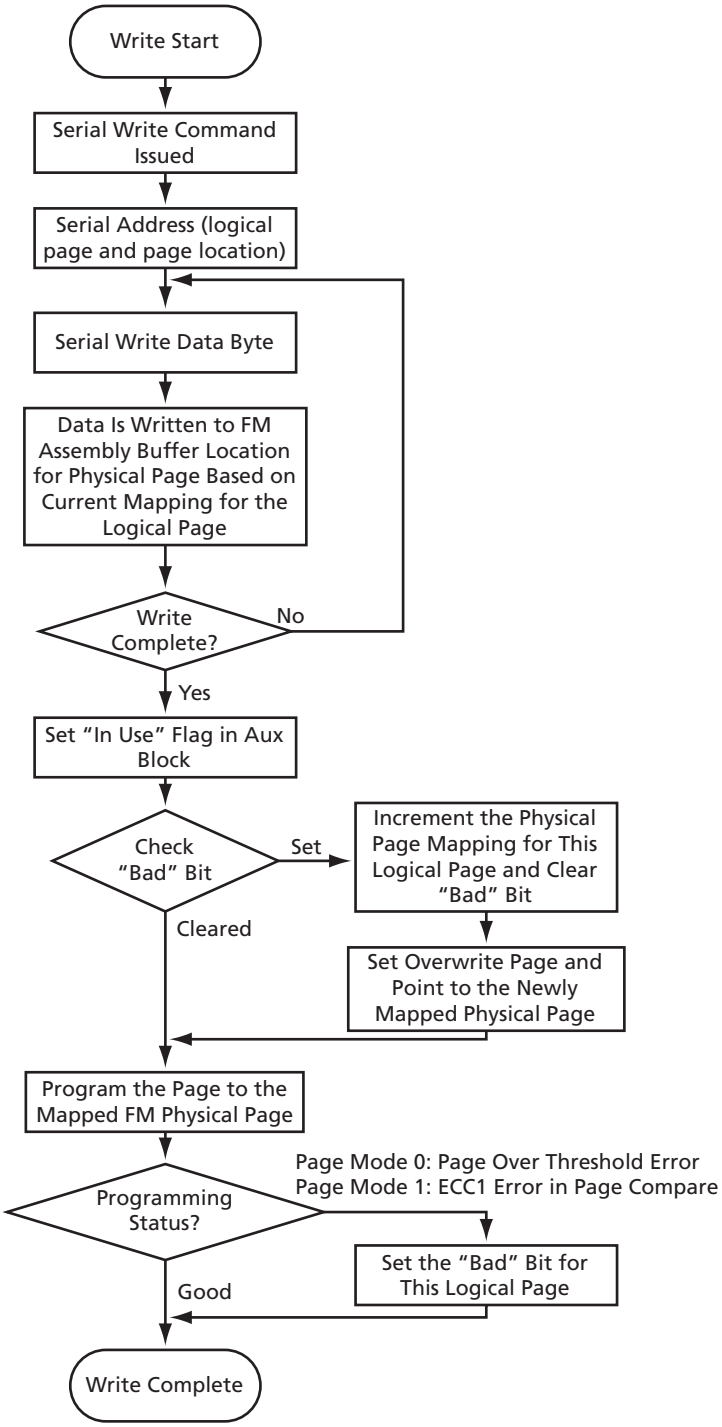


Figure 3-2 · Write Flow Diagram

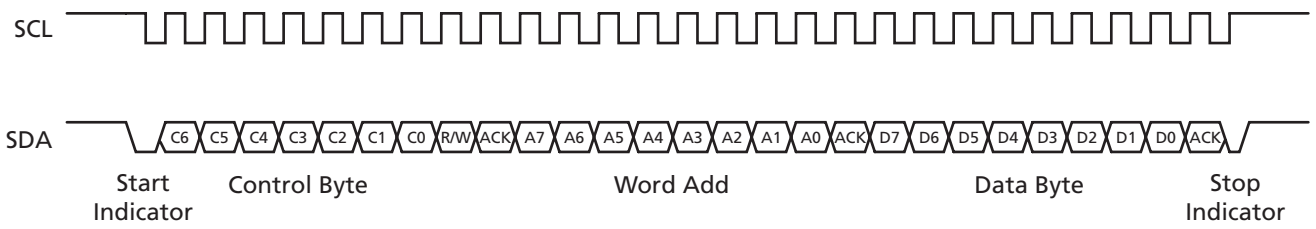
Serial Interface Operation

CoreFMEE responds to a number of serial interface commands. The following subsections outline all the commands that CoreFMEE responds to and the handshaking involved. Note that, due to the use of a serial protocol, the read/write (R/W) flag depends on the context of the master data input. In general, the following conventions are followed:

- SDA is a bidirectional open-drain serial data signal at the pin level of the Fusion device, consisting of the SDA_IN input and SDA_OUT output, the direction of which is controlled by the SDA_CTL signal. The open drain should be implemented using a tristate buffer.
- Data (SDA) is latched at the destination on the rising edge of SCL. Data is sent on the falling edge of SCL.
- A data transition while SCL is HIGH is considered either a start or a stop indicator.
- A start indicator is defined as a HIGH to LOW transition of SDA while SCL is HIGH.
- A stop indicator is defined as a LOW to HIGH transition of SDA while SCL is HIGH.
- SCL is originated at the master and is used by all slave devices.
- SCL is not a continuous running clock and is only active when the master has commands for a slave.
- The ACK cycle may be driven by either the master or CoreFMEE and is dependent on the command being executed.
- An ACK is performed by putting a 0 on the SDA line during the ACK cycle. A NO ACK is performed by driving a 1 on the SDA line during the ACK cycle.
- CoreFMEE will only perform the duties of a slave device.

Serial Write

A serial write is initiated as shown in Figure 3-3.



Note: R/W = 0 for Write

Figure 3-3 · Serial Write

A serial write is initiated by the master, which asserts a start pulse and puts a control byte on the SDA line. The control byte contains the following data:

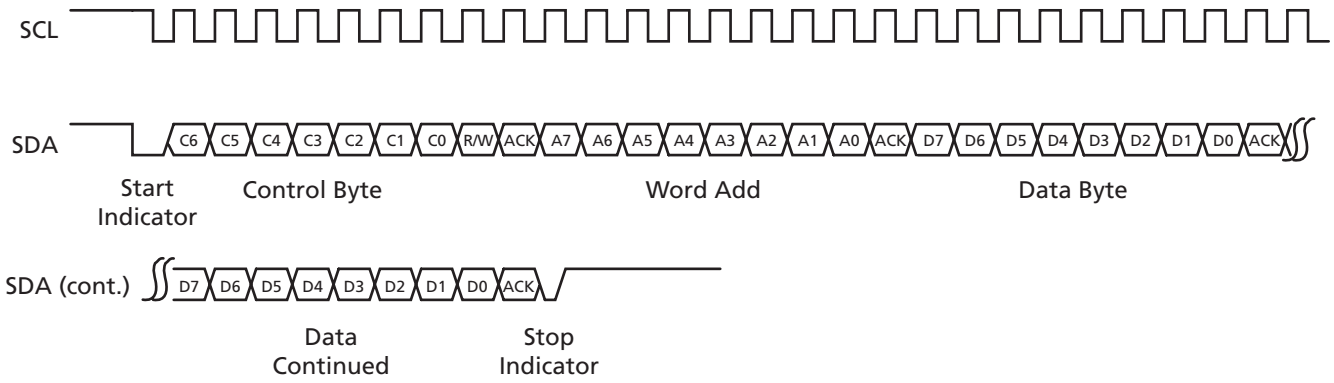
- C6–C3 = '1010'
- C2–C0 = Device address in smaller devices. For larger devices, these bits are used as extended address bits.
- R/W = 0 for write
- ACK = 0 – This is sent by the CoreFMEE device that is being addressed in C2–C0. If the CoreFMEE device does not match the address on C2–C0 with the ADD[2:0] inputs, CoreFMEE does not respond to the command. If for some reason the Fusion Flash memory is unavailable, CoreFMEE will drive a NO ACK on the SDA line, and the master will terminate the transfer with a stop indicator.

After the ACK from CoreFMEE, the master will issue the remainder of the starting address on the SDA line. CoreFMEE will only issue the ACK if it has been granted access to the Flash memory for the write/program sequence. At the end of the address, CoreFMEE is required to issue the ACK for the write address.

After the word address, the master will begin to transmit data bytes on the SDA line. CoreFMEE will acknowledge each byte as it is received. When the master has completed sending the last byte, it will terminate the transfer by executing a stop command. Data transmitted is stored linearly in the Flash memory, starting at the logical address sent by the master.

Page Write

A page write is initiated as shown in Figure 3-4.



Note: R/W = 0 for Write

Figure 3-4 · Page Write

A page write is initiated in exactly the same way that a serial write occurs. The control byte, word address, and first data byte are the same. The only difference is that the master does not issue a stop at the end of the first data byte. It will continue to send data bytes until it has finished sending all bytes for the page.

When the master has finished sending the last byte, it will issue a stop indicator to terminate the transfer. CoreFMEE will ACK each data byte as it is received. Upon terminating the transfer, CoreFMEE will program the data into contiguous locations within Flash memory, as indicated by the logical address.

Page writes are limited to either the size of the Flash memory page (128 bytes) or the end of the Flash memory page. A page write does not have to start at the beginning of the page boundary, but it must terminate at the end of a page boundary (otherwise the address will wrap around to the beginning of the page).

Random Read

A random read is initiated as shown in Figure 3-5.

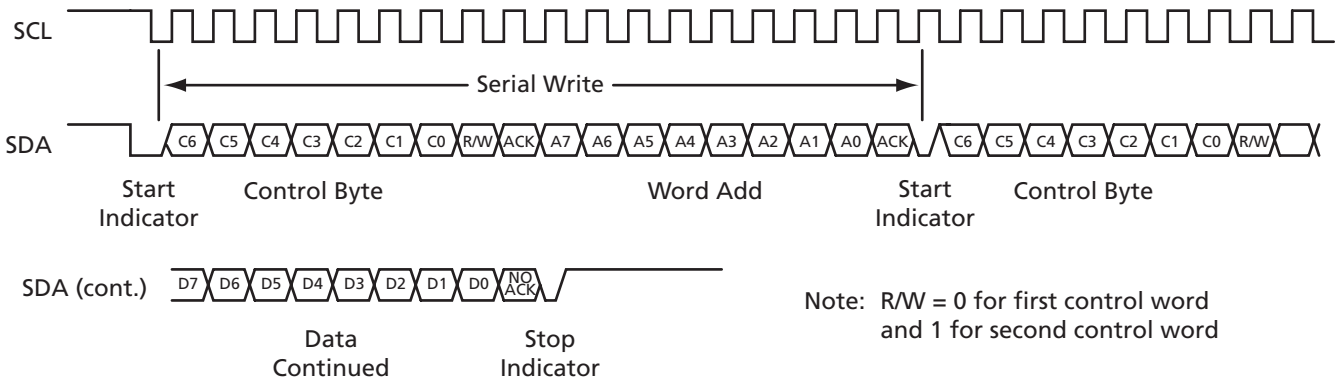


Figure 3-5 · Serial Read

The master initiates a random read by starting to perform a serial write. The sequence for the control byte and word byte are exactly the same as in a serial write. The reason for this is that there is no other mechanism in the serial protocol to load the current read address into CoreFMEE. The way CoreFMEE determines that the command is a read is that the

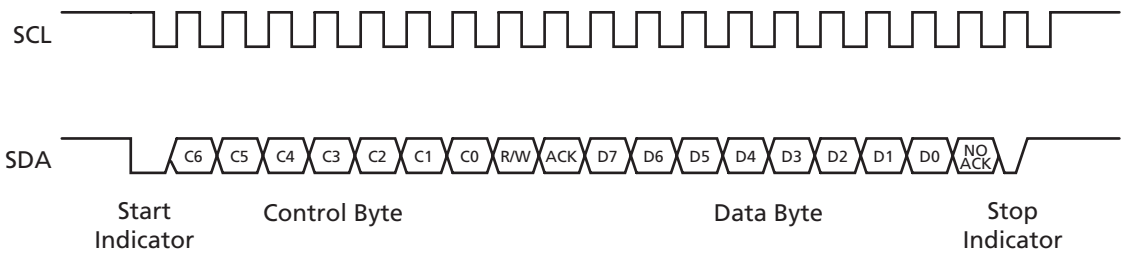
master will issue a second start command after the CoreFMEE ACK for the word address. The second start indicates to CoreFMEE that the command is a serial read starting at the address just received.

After the Word Address/Start sequence, the master will issue another control byte, which will contain the same information as the first control byte, except that the R/W indicator will be set to 1. CoreFMEE will ACK this control word and proceed to send out the byte that was located at the logical address. At the end of the data, the master will issue a NO ACK and stop indicator, terminating the transfer.

The address transmitted for the read is stored in a local address register and is incremented by 1 after the read. Subsequent reads from the master from this base address will not send an address during the command phase.

Current Address Read

A current read operation will read the last byte location read, plus one. If, for example, the master initiated a serial read at address 0x10, a follow-on current address read would retrieve data from address 0x11. If the master performs a read operation that causes the current address to go beyond the end of the logical address range, the address is wrapped around to the first location in the logical address space. Figure 3-6 outlines the interface requirements.



Note: R/W = 1 for Read

Figure 3-6 · Current Address Read

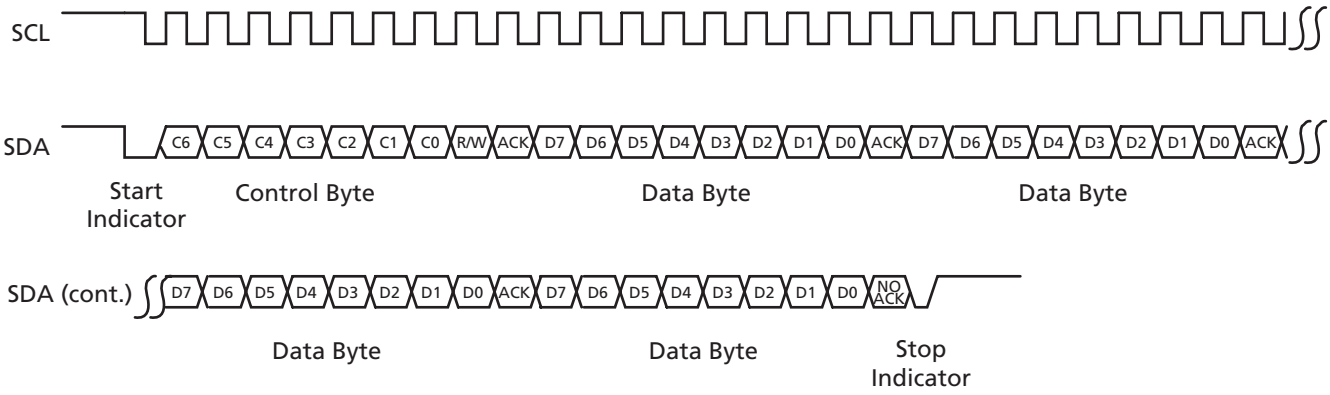
The current address read contains a command byte and a data byte. The command byte is proceeded by a start command; the command byte contains the correct command sequence in C6–C3 and device address in C2–C0, and the R/W bit is set to 1. CoreFMEE will acknowledge the command if it has gained access to the Flash memory.

The data for the logical address will be sent out on the SDA line during the data byte transmission. The master will set NO ACK and the stop bit to terminate the transfer.

CoreFMEE will increment the local logical read address for any subsequent reads.

Sequential Read

A sequential read is initiated as shown in Figure 3-7.



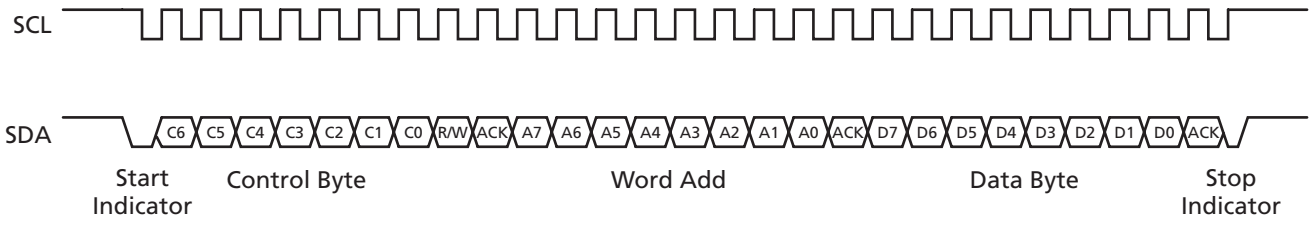
Note: R/W = 1 for Read

Figure 3-7 · Sequential Address Read

A sequential read is continuation of a current address read. The sequential read differs from the current address read in that the master does not terminate the data read after the first byte. When this occurs, CoreFMEE must continue to send data to the master until the master sends a NO ACK and stop indicator. When the end of a 128-byte logical page is reached, the address will wrap back to the beginning of the page. CoreFMEE will increment its internal address to coincide with the logical address the master is requesting.

Write Protect Operation

A write protect operation is initiated as shown in Figure 3-8.



Note: R/W = 0 for Write

Figure 3-8 · Write Protect

The fields for a write protect operation are similar to those used in a write operation. The differences are that C6–C3 of the control byte are set to '0110' and the word address and data byte fields are “don’t care.”

Write protection is supported in accordance with Table 3-3.

Table 3-3 · Write Protect Operations

WP_MODE Generic	WP Input	WP Internal Register	Part of Array Protected
0	1	X	Full Array
0	0	X	None
1	1	X	Full Array
1	0	Not Programmed	None
1	0	Programmed	Full Array

Notes:

1. *'X' indicates "don't care."*
2. *Once a portion of the logical address space has been write-protected, there is no provision for changing the write protections.*
3. *If a write operation is attempted to a write-protected portion of the Fusion Flash memory, CoreFMEE will ACK the operation but no writes will occur to the Flash memory.*

Acknowledge Operation for Write Protect

When CoreFMEE is in software write protect mode (WP_MODE generic set to 1), it provides different responses to commands depending on its write protect state. Table 3-4 outlines the operations.

Table 3-4 · Software Write Protect Operations

Start Command	R/W	Write Protect State	ACK / NO ACK from CoreFMEE	Action from CoreFMEE
WP Input Connected to Logic 0				
1010	R	X	ACK	Read CoreFMEE
1010	W	Programmed	ACK	CoreFMEE write-protected
1010	W	Not programmed	ACK	Write to CoreFMEE
0110	R	Programmed	NO ACK	Indicates WP register programmed
0110	R	Not programmed	ACK	Indicates WP register not programmed
0110	W	Programmed	NO ACK	Already programmed
0110	W	Not programmed	ACK	Sets WP register state
WP Input Connected to Logic 1				
1010	R	X	ACK	Read CoreFMEE
1010	W	Programmed	ACK	CoreFMEE write-protected
1010	W	Not programmed	ACK	CoreFMEE write-protected
0110	R	Programmed	NO ACK	Indicates WP register programmed
0110	R	Not programmed	ACK	Data is “don’t care”
0110	W	Programmed	NO ACK	Already programmed
0110	W	Not programmed	ACK	Sets WP register state

Note: ‘X’ indicates “don’t care.”

Acknowledge Polling

Once the internal circuitry has started a program sequence for the Flash memory and no other operations are allowed, the master can perform acknowledge polling. The master sends a start condition followed by a device address word. If CoreFMEE is available for read/write operations, it will acknowledge the command. Otherwise, the master will assume that CoreFMEE is still busy performing the write/program.

Serial Interface Reset

After an interruption in protocol, power loss, or system reset, CoreFMEE can be reset by following these steps:

1. Clock up to nine cycles.
2. Look for SDA HIGH in each cycle while SCL is HIGH.
3. Create a start condition.

One caveat to the reset operation is what happens when the Flash memory is in the middle of a program operation. The Flash memory cannot abort the program (without unrecoverable results), so the interface will be held off after a reset until the Flash memory finishes the program.

Flash Memory Arbitration

The COREFMEE_SERIAL_SLAVE block will make a request to the COREFMEE_NVM_INT block when it sees an address during the control period of the serial transfer that matches the internal address. If the block does not actively receive the FM_ACK signal 1¾ SCL clock periods later, the COREFMEE_SERIAL_SLAVE block will NO ACK the command. Upon the request made by the COREFMEE_SERIAL_SLAVE block to the COREFMEE_NVM_INT block, the COREFMEE_NVM_INT block will activate the FM_REQ signal. When the COREFMEE_NVM_INT block actively receives the FM_ACK signal, it sends an acknowledge to the COREFMEE_SERIAL_SLAVE block.

The COREFMEE_SERIAL_SLAVE block will make an active request to the COREFMEE_NVM_INT block as long as the serial interface is active; it will cease this request upon the receipt of a STOP indicator. The arbitration logic within the COREFMEE_NVM_INT block will continue to hold the FM_REQ signal active after the COREFMEE_SERIAL_SLAVE block ceases its request if the COREFMEE_SERIAL_SLAVE block has been performing a write operation or if its serial protection has been set. In either case, the COREFMEE_NVM_INT block will perform a program operation, and the arbitration logic will keep the FM_REQ signal active until the program operation(s) have completed. The arbitration logic within the COREFMEE_NVM_INT block will not acknowledge another request from the COREFMEE_SERIAL_SLAVE block until the program operation has finished.

4

Testbench Operation and Modification

Verification Testbench

Included with the release of CoreFMEE is a verification testbench that verifies operation of the CoreFMEE macro. A simplified block diagram of the verification testbench is shown in Figure 4-1.

The verification test suite includes a verification testbench, which exercises the CoreFMEE design and scripts to run the testbench for different combinations of generics/parameters. The testbench instantiates and interconnects the design under test (DUT), which is the CoreFMEE macro and the Fusion Flash memory behavioral model. Note that to fully verify the CoreFMEE macro, regression scripts are included that run the macro using all legal combinations of generics/parameters.

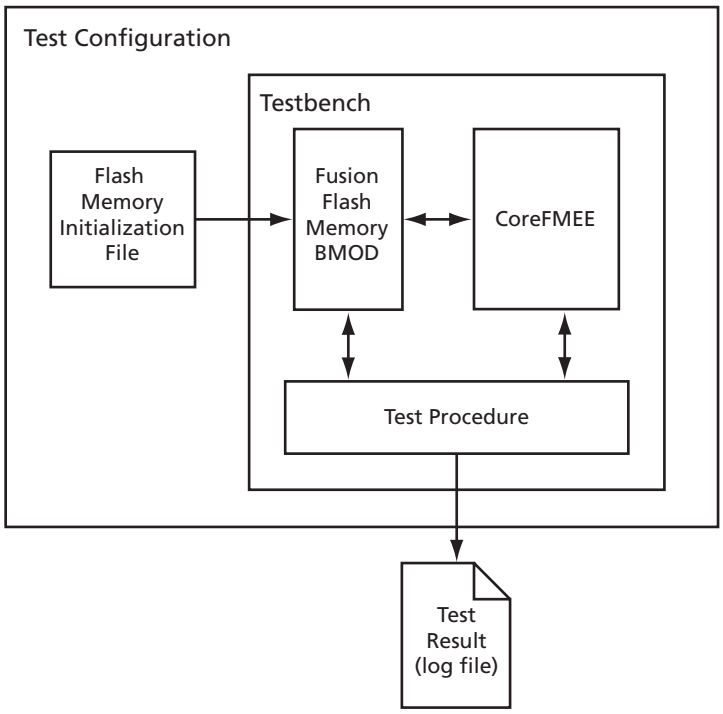


Figure 4-1 · CoreFMEE Verification Testbench

The source code for the verification testbench is available only with the CoreFMEE RTL release.

Verification Tests

CoreFMEE is verified through a number of tests that exercise CoreFMEE through the external interface. The CoreFMEE verification testbench uses the Fusion Flash memory behavioral model to simulate the behavior of the Flash memory in Fusion devices. The memory behavioral model is provided (as a library cell) as part of Libero IDE for Actel Fusion products.

The verification testbench includes test procedures to check the following FMEE operations:

- Serial Write, using single-byte and page mode
- Serial Read, using random, current, and sequential modes
- Write Protect and Unprotect, both hardware and software
- Endurance extension using ECC errors to advance the logical page pointer
- Endurance extension using overthreshold flag to advance the logical page pointer
- Rejection of invalid serial commands
- Address matching on the ADD[2:0] inputs

Simple Application Testbench

An example user testbench is included with the Evaluation, Obfuscated, and RTL releases of CoreFMEE. The user testbench is provided in pre-compiled Model*Sim* format and in RTL source code for all releases (Evaluation, Obfuscated, and RTL) for you to examine and modify to suit your needs. The source code for the user testbench is provided to ease the process of integrating the CoreFMEE macro into your design and verifying according to your own custom needs. A block diagram of the user testbench is shown in Figure 4-2.

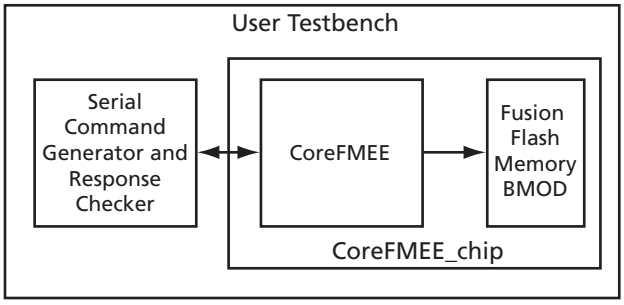


Figure 4-2 · CoreFMEE User Testbench

The user testbench includes a simple example design that serves as a reference for users who want to implement their own designs.

The testbench for the example user design implements a subset of the functionality tested in the verification testbench, described in "Verification Testbench" on page 29. Conceptually, as shown in Figure 4-2, CoreFMEE and the Fusion Flash memory BMOD are instantiated into a chip-level wrapper, which is instantiated along with an external two-wire serial master, which generates serial commands to read and write the CoreFMEE.

Once you have familiarized yourself with the HDL source code for the user testbench, you may wish to customize it, recompile, and run the simulation, as described in the "Implementation Hints" on page 31.

5 Implementation Hints

This chapter provides various hints to ease the process of implementation and integration of CoreFMEE into your own design.

Usage with Internal Flash Memory

Proper operation of the CoreFMEE design requires the use of the Fusion Flash memory. The Fusion Flash memory is an integral part of the CoreFMEE design—CoreFMEE will not function properly without it. CoreFMEE provides a transparent interface to the Flash memory that should not be modified. CoreFMEE should be connected to the Flash memory as shown in the example designs provided. If the interface is altered, it is likely that CoreFMEE will cease to function properly.

It is anticipated that CoreFMEE will be used as an interface for components external to the Actel Fusion device. Components internal to the Fusion device will see the best performance if they use a direct interface to the internal Flash memory.

The Fusion Flash memory used with CoreFMEE can be programmed through the FMEE interface, or it can be pre-programmed independently from the FPGA fabric by use of the FlashPro software and hardware (refer to the *FlashPro User's Guide* for details on how to program the Flash memory within Fusion devices).

The Fusion Flash memory program operation always writes 128 bytes of data, regardless of the actual write size desired. For a given page (128 bytes) being written, if only one of the 128 bytes was changed by the user, the other 127 bytes will be written again with the unchanged value. It is best to keep this in mind when writing to the Flash memory. Though the purpose of this macro is to maximize the lifetime of the Flash memory by using multiple physical pages to store each logical page, the endurance (lifetime) of the Flash memory will be maximized (especially for low values of the ENDURANCE generic) if the user minimizes single-location writes (e.g., write all of the desired locations for a given page using a page write instead of multiple single writes). Refer to the *Fusion Family of Mixed-Signal FPGAs* datasheet for information on the Flash memory endurance specifications.

A

VHDL Testbench Support Routines

The user application testbench for the CoreFMEE macro makes use of VHDL procedures (or tasks in the Verilog version). The main support procedures that are useful in developing customized testbenches are described in this appendix.

The support procedures are incorporated into the testbench procedure to reduce the number of signals that need to be passed in the procedure calls.

```
procedure do_eeprom (   -- note that this is an overloaded procedure
  constant operation  : in    t_eeprom_accesses;
  constant dev_addr   : in    natural;
  constant byte_addr  : in    natural;
  constant data_in    : in    t_eeprom_data;
  constant check_data : in    boolean := false;  -- only for reads
  variable data_out   : out   t_eeprom_data;
  variable ack        : inout boolean)

procedure do_eeprom (
  constant operation    : in  t_eeprom_accesses;
  constant dev_addr     : in  natural;
  constant byte_addr    : in  natural;
  constant data_in      : in  t_eeprom_data;
  constant check_data   : in  boolean := false;  -- only for reads
  variable data_out     : out t_eeprom_data;
  chk_timeout           : in  boolean := true;
  variable chk_prot_ack : out boolean)

procedure poll_acknowledge (
  constant dev_addr     : in natural)

procedure write_page (
  constant page_addr  : in natural;
  constant ee_add     : in natural)

procedure read_page (
  constant page_addr : in natural;
  constant ee_add    : in natural)
```

B

Product Support

Actel backs its products with various support services including Customer Service, a Customer Technical Support Center, a web site, an FTP site, electronic mail, and worldwide sales offices. This appendix contains information about contacting Actel and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From Northeast and North Central U.S.A., call **650.318.4480**
From Southeast and Southwest U.S.A., call **650. 318.4480**
From South Central U.S.A., call **650.318.4434**
From Northwest U.S.A., call **650.318.4434**
From Canada, call **650.318.4480**
From Europe, call **650.318.4252** or **+44 (0) 1276 401 500**
From Japan, call **650.318.4743**
From the rest of the world, call **650.318.4743**
Fax, from anywhere in the world **650.318.8044**

Actel Customer Technical Support Center

Actel staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions. The Customer Technical Support Center spends a great deal of time creating application notes and answers to FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Actel Technical Support

Visit the Actel Customer Support website (www.actel.com/custsup/search.html) for more information and support. Many answers available on the searchable web resource include diagrams, illustrations, and links to other resources on the Actel web site.

Website

You can browse a variety of technical and non-technical information on Actel's home page, at www.actel.com.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. Several ways of contacting the Center follow:

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is tech@actel.com.

Phone

Our Technical Support Center answers all calls. The center retrieves information, such as your name, company name, phone number and your question, and then issues a case number. The Center then forwards the information to a queue where the first available application engineer receives the data and returns your call. The phone hours are from 7:00 A.M. to 6:00 P.M., Pacific Time, Monday through Friday. The Technical Support numbers are:

650.318.4460
800.262.1060

Customers needing assistance outside the US time zones can either contact technical support via email (tech@actel.com) or contact a local sales office. Sales office listings can be found at www.actel.com/contact/offices/index.html.

Index

For more information about Actel's products, visit our website at http://www.actel.com

Actel Corporation *• 2061 Stierlin Court • Mountain View, CA 94043 USA*
Customer Service: 650.318.1010 • Customer Applications Center: 800.262.1060

Actel Europe Ltd*. • River Court, Meadows Business Park • Station Approach, Blackwater • Camberley Surrey GU17 9AB • United Kingdom*
Phone +44 (0) 1276 609 300 • Fax +44 (0) 1276 607 540

Actel Japan *• EXOS Ebisu Bldg. 4F • 1-24-14 Ebisu Shibuya-ku • Tokyo 150 • Japan*
Phone +81.03.3445.7671 • Fax +81.03.3445.7668 • www.jp.actel.com

Actel Hong Kong *• Suite 2114, Two Pacific Place • 88 Queensway, Admiralty Hong Kong*
Phone +852 2185 6460 • Fax +852 2185 6488 • www.actel.com.cn

50200098-0 /3.07